The purpose of this discussion is to estimate the operations count for a single GEONS propagation and update cycle, assuming four MMS satellites processing standard GPS and remote/local and remote/remote crosslink tracking data. Determining operations counts for large software (such as GEONS) is not straight-forward. The main uncertainties are the platform on which the counts are needed and the compiler used. Different platforms have different instruction sets, which can be executed in a different number of cycles. The options used to compile the C code will impact the number of counts, assuming higher optimization levels would produce more efficient assembly code. The compiler brand will also affect the number of counts, since one manufacturer may produce a more robust compiler than another manufacturer.

Ideally, the program would be compiled into assembly code, where the opcodes could be counted. The manufacturer would have a listing of the times each opcode would require to execute, and an estimate of the number of counts and time used for a particular code segment could be determined. Initially, this would be a time consuming project, but once the opcodes for the specific platform has been collected, subsequent determination of counts would only require re-compilation of the source code.

Another method that would be less time consuming (initially) would use the time a code segment needed to execute in and the number of floating point operations per second the platform can support to compute the number of counts. This is the approach examined in this discussion.

Floating-Point Performance

The computer system used is *jade*, which is a Dell 1.2 Giga-Hertz (GHz) single-processor machine running Linux. It is assumed that the clock used to provide the timing information is in working order (unlike the clock on old *qui-gon*). The benchmarks are based on the Livermore FORTRAN Kernels written in C (http://www.aip.org/cip/articles.htm). The benchmarks were compiled using both full optimization and full debug (unoptimized) modes using the GNU C compiler. Each benchmark was run ten times to obtain a statistical sampling. The geometric mean of the floating point operations were used per the Reference. Figure 1 shows the number of floating point operations for each sample. Table 1 summarizes the ensemble statistics.

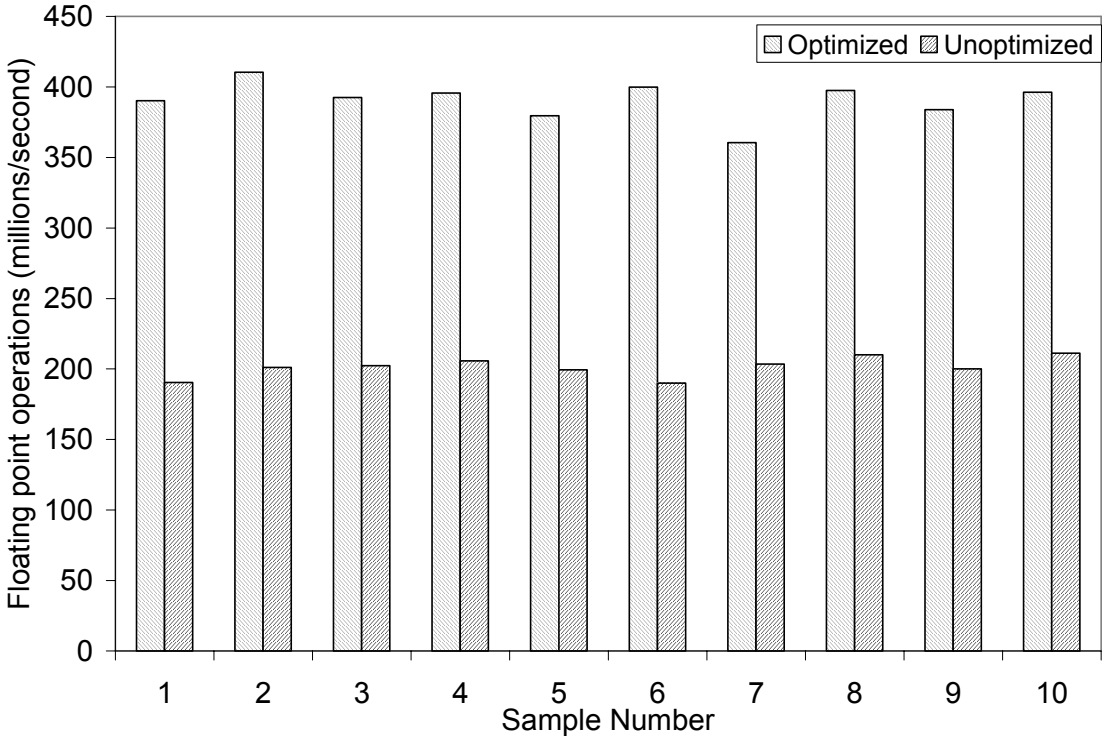
Table 1 Millions of Floating Point Operations per Second Based on the Livermore FORTRAN Kernel Benchmarks for the Jade Machine

	Optimized	Unoptimized
Average	391	201
Standard deviation	14	7.1
Root-Mean-Square (RMS)	391	202
Minimum	361	190
Maximum	410	211
3-Sigma Confidence Interval	(374,408)	(193,210)

The optimized benchmarks produced an average of 391 million floating point operations per second (MFlops), while the unoptimized benchmarks produced 201 MFlops.

It should be noted that other processes were running while the benchmarks were run, so additional uncertainties due to page swapping and other system background jobs could affect the benchmark results.

Figure 1 Millions of Floating Point Operations per Second Based on the Livermore FORTRAN Kernel Benchmarks



Profiling GEONS

The GNU profiler was used to determine the time consumed in the routine EstimateState (including the time spent in routines called by EstimateState). Two GEONS executables were made: one using full optimization and the other using full debug. Ten runs were made for each compilation mode to obtain a statistical sample. The time slice used by the profiler is 0.01 seconds. The total time spent in EstimateState was divided by the total number of times it was called to obtain an average time spent per call (TSPC). This value was then multiplied by the MFlops values as an estimate of the number of floating point operations used per call (MFlopc).

$$MFlopc = \left(\frac{Flop}{time}\right)\cdot\left(\frac{time}{calls}\right)$$
$$= MFlops \cdot TSPC$$

Case 1: GPS on local only, remote/local and remote/remote crosslink

This case processes GPS measurements every minute for the local satellite only and remote/local and remote/remote crosslink measurements every 6 minutes throughout the orbit. The slow cycle is one minute. Both optimized and unoptimized executables are examined. Table 2 summarizes the results.

Table 2 Millions of Floating Point Operations per EstimateState Call for Case 1

	Optimized	Unoptimized
Total Time (seconds)	178.08	352.94
Number of Times Called	50410	50410
Average MFlopc	1.380	1.410
3-Sigma Confidence Interval (MFlopc)	(1.320,1.440)	(1.348,1.472)

Note that even though the unoptimized code took approximately twice as long to run as the optimized code, the average MFlopc values are similar since the optimized MFlops is approximately twice as large as the unoptimized value. The time used to execute all ten samples was 1012 seconds for the unoptimized GEONS and 433 seconds for the optimized version.

As an aside, the routines where most of the CPU time was spent are Time_Minus_Time, MatrixProduct, and PropagateCovariance using the optimized GEONS executable:

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 34.88    151.62   151.62 28332100     0.01     0.01  Time_Minus_Time
 19.17    234.95    83.33 10975250     0.01     0.01  MatrixProduct
 17.72    311.96    77.01    50400     1.53     1.73  PropagateCovariance
  6.07    338.34    26.38   604800     0.04     0.05  g2plan
```

MatrixProduct consumed the majority of the CPU time for the unoptimized GEONS executable:

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 58.44    593.15   593.15 10975250     0.05     0.05  MatrixProduct
  7.05    664.71    71.56    50400     1.42     1.83  PropagateCovariance
  7.00    735.74    71.03   604800     0.12     0.12  g2plan
```

Case 2: GPS on local and remote, remote/local and remote/remote crosslink

This case processes GPS measurements every minute for all the satellites and remote/local and remote/remote crosslink measurements for 10 minutes every hour. The affect of changing the slow cycle from 60 seconds to 10 seconds is also examined when processing GPS data only. Since the previous section demonstrated that the optimized and unoptimized executables produce similar MFlopc values, only the optimized executable is examined for this case. Table 3 summarizes the results.

Table 3 Millions of Floating Point Operations per EstimateState Call for Case 2

	GPS-only	GPS-only	GPS/Crosslink
Slow Cycle (seconds)	60	10	10
Total Time (seconds)	234.14	696.86	1336.45
Number of Times Called	50410	302410	302410
Average MFlopc	1.814	0.900	1.726
3-Sigma Confidence Interval (MFlopc)	(1.736,1.893)	(0.861,0.939)	(1.651,1.802)

The 10-second GPS-only case produced an average MFlopc of 0.9, which is approximately half of the average MFlopc for the 60-second GPS-only case. This indicates that the propagation-only process (which occurs 5 out of 6 times) is less expensive than the update process.

Case 3: Single Satellite GPS and remote/local crosslink

This case examines Case 2 for a single MMS satellite using a 10-second slow cycle. Three precise ephemeris files are used for the remaining satellites in the constellation. Table 4 summarizes the results.

Table 4 Millions of Floating Point Operations per EstimateState Call for Case 3

Total Time (seconds)	155.53
Number of Times Called	302410
Average MFlopc	0.201
3-Sigma Confidence Interval (MFlopc)	(0.192,0.210)

Case 4: Single Satellite GPS and remote/local crosslink

This case processes GPS and cross-link measurements every minute for a single Aurolites satellite using a 30-second slow cycle. Three precise ephemeris files are used for the remaining satellites in the constellation. A 2-day definitive span is used. An additional 1-day prediction span is also examined. Table 5 summarizes the results.

Table 5 Millions of Floating Point Operations per EstimateState Call for Case 4

	2 day definitive only	1 day prediction
Total Time (seconds)	20.06	22.85
Number of Times Called	57610	86410
Average MFlopc	0.136	0.103
3-Sigma Confidence Interval (MFlopc)	(0.130,0.142)	(0.099,0.108)